Where Do Heuristics Come From ? Part 3

Robert C. Holte
Computing Science Department
University of Alberta

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Outline

Part 1: Introduction

Part 2: Details

Part 3: Pattern Database Enhancements

- Taking the maximum of two or more PDBs
- Compression and Dovetailing of PDBs
- Additive PDBs
- Customized PDBs
- Multiple Lookups in One PDB

Bonus! – Related Algorithm (CFPD)

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Max'ing Multiple Heuristics

- Given heuristics h1 and h2 define

  h(s) = max (h1(s), h2(s))

- Preserves key properties:
  - lower bound
  - consistency

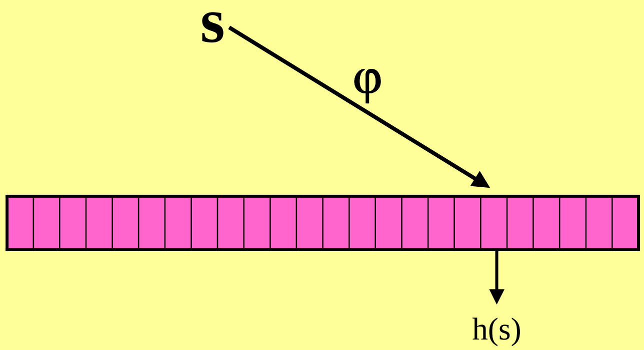
ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Question

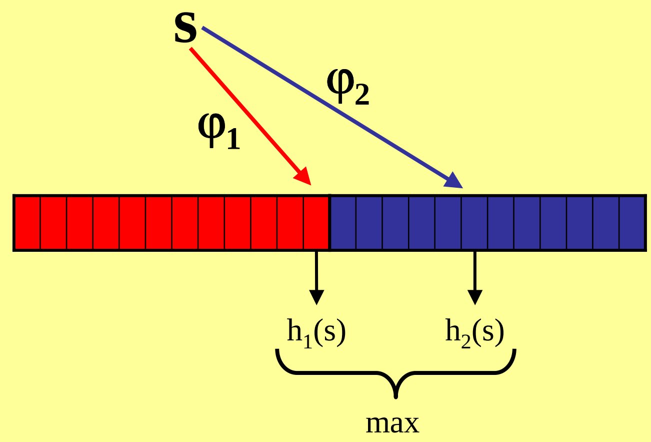
- Given a fixed amount of memory, M, which gives the best heuristic ?
  - 1 pattern database (PDB) of size M
  - max'ing 2 PDBs of size M/2
  - max'ing 3 PDBs of size M/3
  - etc.

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

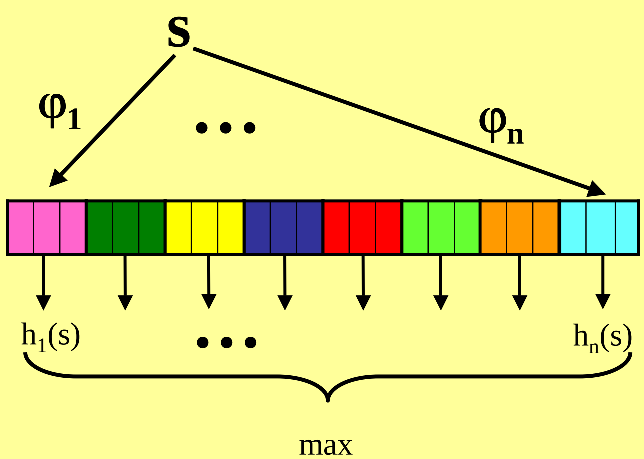
1 large pattern database



2 half-size pattern databases



Many small pattern databases



Rubik's Cube*

PDB Size	n	Nodes Generated
13,305,600	8	2,654,689
17,740,800	6	**2,639,969**
26,611,200	4	3,096,919
53,222,400	2	5,329,829
106,444,800	1	**61,465,541**

* "easy" problems

Summary

State Space	Best n	Ratio
(3x3)-puzzle	**10**	**3.85**
9-pancake	**10**	**8.59**
(8,4)-Topspin (3 ops)	**9**	**3.76**
(8,4)-Topspin (8 ops)	**9**	**20.89**
(3x4)-puzzle	**21+**	**185.5**
Rubik's Cube	**6**	**23.28**
15-puzzle (additive)	**5**	**2.38**
24-puzzle (additive)	**8**	**1.6 to 25.1**

$$\text{RATIO} = \frac{\text{\#nodes generated using } \textbf{one} \text{ PDB of size M}}{\text{\#nodes generated using } \textbf{n} \text{ PDBs of size M/n}}$$

Rubik's Cube CPU Time

#PDBs	Nodes Ratio	Time Ratio
8	**23.15**	**12.09**
6	**23.28**	**14.31**
4	**19.85**	**13.43**
2	**11.53**	**9.87**
1	**1.00**	**1.00**

time/node is 1.67x higher using **six** PDBs

Techniques for Reducing the Overhead of Multiple PDB lookups

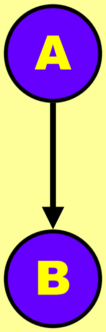
Early Stopping

IDA* depth bound = 7
g(s) = 3
⇒ Stop doing PDB lookups as soon as h > 4 is found.

Might result in extra IDA* iterations

$PDB_1(s) = 5 \Rightarrow$ next bound is 8
$PDB_2(s) = 7 \Rightarrow$ next bound is 10

Consistency-based Bounding



$PDB_1(A) = 1$
$PDB_2(A) = 7$

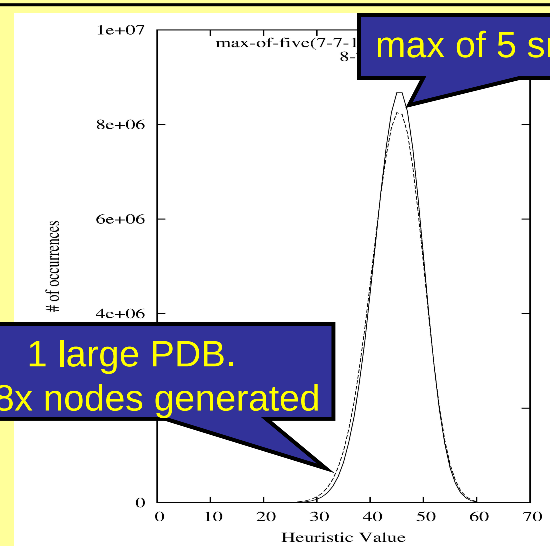
Because of consistency:
$PDB_1(B) \leq 2$
$PDB_2(B) \geq 6$
$\Rightarrow$ No need to consult PDB_1

Experimental Results

- 15-puzzle, five additive PDBs (7-7-1)
  - Naïve: 0.15 secs
  - Early Stopping: 0.10 secs
- Rubik's Cube, six non-additive PDBs
  - Naïve: 27.125 secs
  - Early Stopping: 8.955 secs
  - Early Stopping and Bounding: 8.836 secs

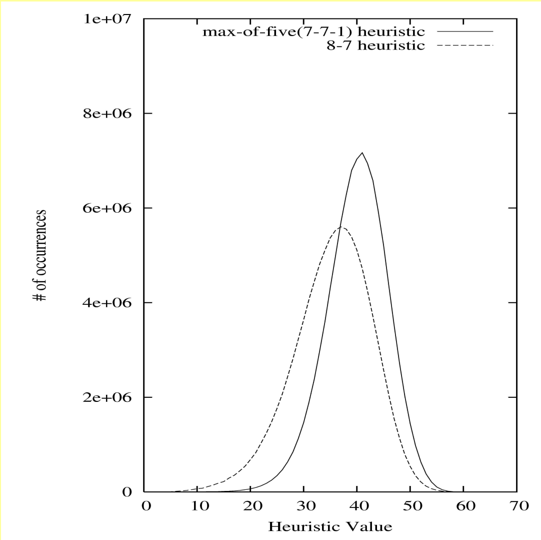
Why Does Max'ing Speed Up Search ?

Static Distribution of Heuristic Values



15-puzzle, 100M states.

Runtime Distribution of Heuristic Values



Example of Max Failing

Depth Bound	h1	h2	max(h1,h2)
8	19	17	10
9		36	16
10	59	78	43
11		110	53
12	142	188	96
13		269	124
14	440	530	314
15		801	400
16	1,045	1,348	816
17		1,994	949
18	2,679	3,622	2,056
19		5,480	2,435
20	1,197	1,839	820
TOTAL	**5,581**	**16,312**	**8,132**

Squeezing More into Memory

Approaches

- Compress an individual Pattern Database
  - Lossless compression
  - Lossy compression must maintain admissibility
  - Allows you to
    - use a PDB bigger than will fit in memory
    - use multiple PDBs instead of just one
- Merge two PDBs into one the same size
  - Culberson & Schaeffer's dovetailing

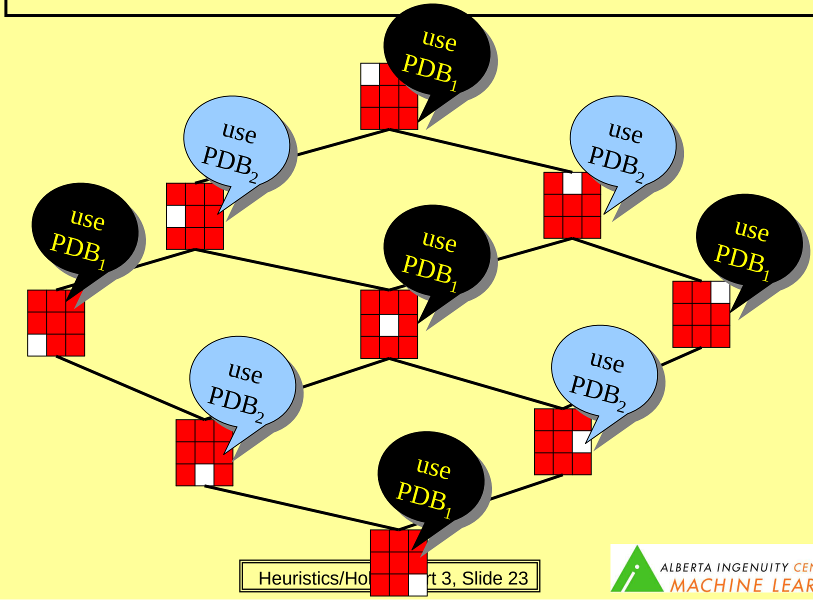
Compression Results

- 16-disk 4-peg TOH, PDB based on 14 disks
  - No compression: 256Megs memory, 14.3 secs
  - lossless compression: 256k memory, 23.8 secs
  - Lossy compression: 96Megs, 15.9 secs
- 15-puzzle, additive PDB triple (7-7-1)
  - No compression: 537Megs memory, 0.069 secs
  - Lossy compression, **two** PDB triples
    537Megs memory, 0.021 secs

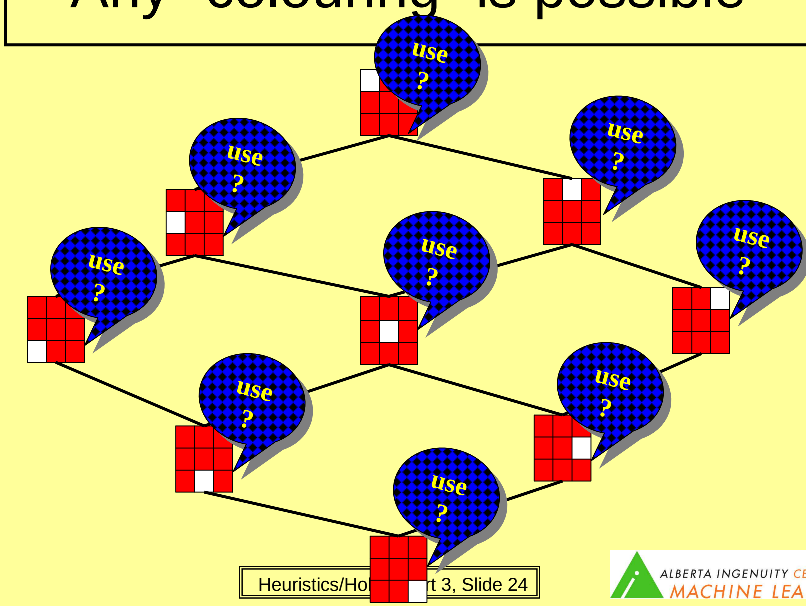
Dovetailing

- Given 2 PDBs for a state space construct a hybrid containing some entries from each of them, so that the total number of entries is the same as in one of the originals.
- The hope: almost as good as max, but only half the memory.

Dovetailing based on the blank



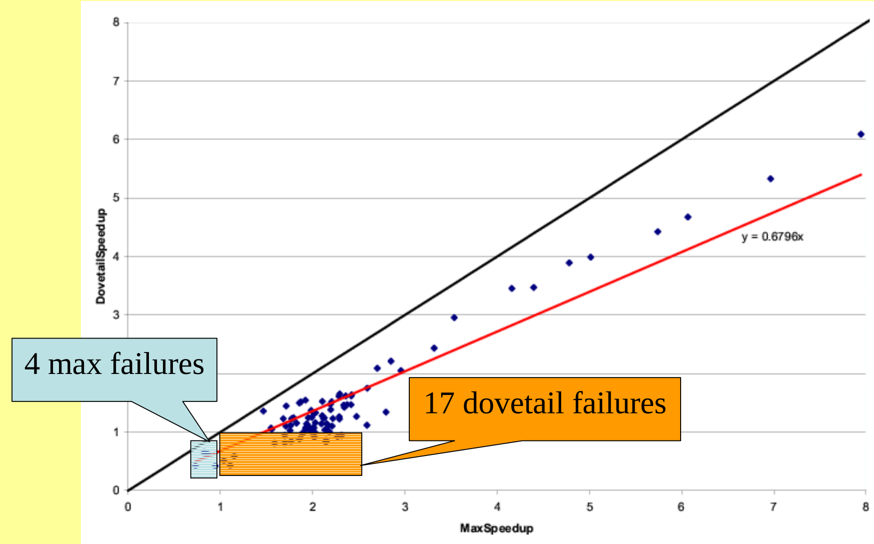
Any "colouring" is possible



Dovetailing – selection rule

- Dovetailing requires a rule that maps each state, s, to one of the PDBs. Use that PDB to compute h(s).
- Any rule will work, but they won't all give the same performance.
- Intuitively, strict alternation between PDBs expected to be almost as good as max.

Dovetailing compared to Max'ing



Experimental Results

- Culberson & Schaeffer (1994):
  - Dovetailing two PDBs reduced #nodes generated by a factor of 1.5 compared to using either PDB alone
- Holte & Newton (unpublished):
  - Dovetailing halved #nodes generated on average

How to generalize Dovetailing to any abstractions of any space ?

A Partial-Order on Domain Abstractions

- Easy to enumerate all possible domain abstractions

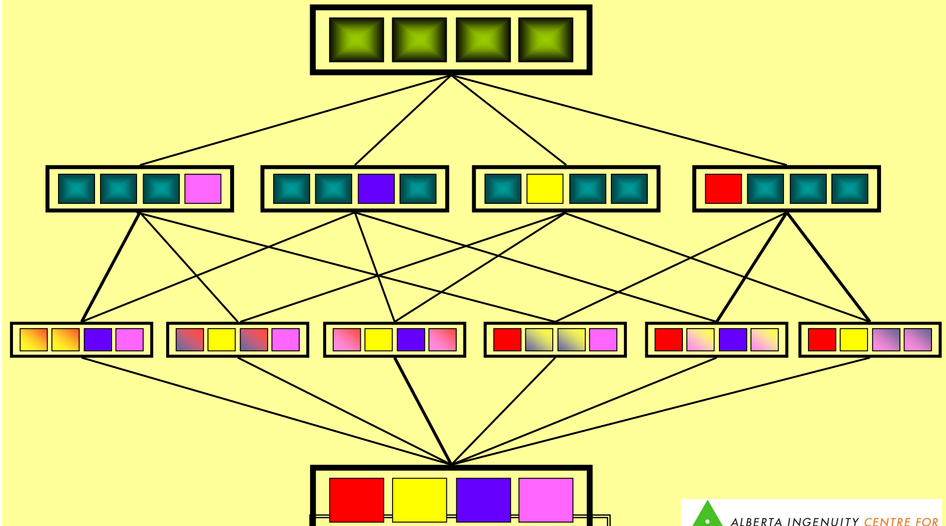
Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank

- and to define a partial-order on them, e.g.

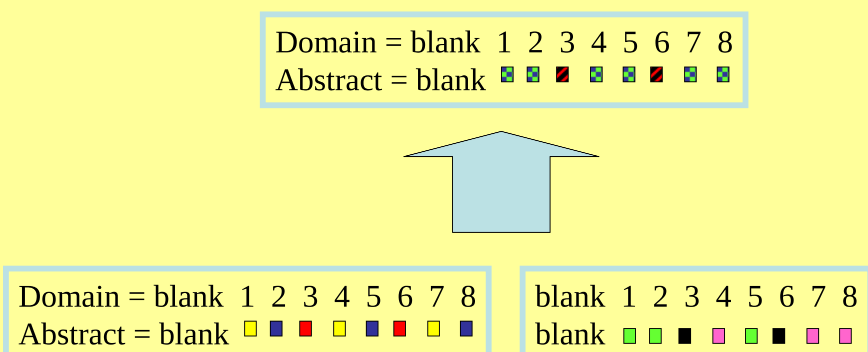
Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank

is "more abstract" than the domain abstraction above.

Lattice of domain abstractions



The "LCA" of 2 Abstractions



LCA = least-abstract common abstraction

General Dovetailing

- Given PDB_1 and PDB_2 defined by φ_1 and φ_2
- Find a common abstraction φ of φ_1 and φ_2
- Because it is a common abstraction there exist $\underline{\varphi}_1$ and $\underline{\varphi}_2$ such that $\underline{\varphi}_1 \varphi_1 = \underline{\varphi}_2 \varphi_2 = \varphi$
- For every pattern, p, defined by φ, set SELECT[p] = φ_1 or φ_2
- Keep every entry (p_k,h) from PDB_i for which SELECT[$\underline{\varphi}_i(p_k)$]=i.
- Given state s
  1. Φ_S = SELECT[φ(s)]
  2. h(s) = PDB[Φ_S (s)]

Additive Pattern Databases

Adding instead of Max'ing

- Under some circumstances it is possible to add the values from two PDBs instead of just max'ing them <u>and still have an admissible heuristic.</u>

- This is advantageous because*

$$h_1(s) + h_2(s) \geq \max(h_1(s), h_2(s))$$

* but see slide "Compared to Max'ing"

Manhattan Distance Heuristic

For a sliding-tile puzzle, Manhattan Distance looks at each tile individually, counts how many moves it is away from its goal position, and adds up these numbers.

	1
2	3

goal

3	
1	2

state s

MD(s) = 2 + 1 + 2 = 5

M.D. as Additive PDBs (1)

$$\varphi_1(x) = \begin{cases} \textbf{x} & \textbf{if} \quad \textbf{x = 1} \\ \textbf{blank} & \textbf{otherwise} \end{cases}$$

	1

φ_1(goal)

1	

φ_1(s)

$PDB_1[\varphi_1(s)] = 2$

$MD(s) = PDB_1[\varphi_1(s)]$
$+ PDB_2[\varphi_2(s)]$
$+ PDB_3[\varphi_3(s)]$

In General…

Partition the tiles in groups, $G_1, G_2, \ldots G_k$

$$\varphi_i(x) = \begin{cases} \mathbf{x} & \textbf{if} \;\; \mathbf{x} \in G_i \\ \textbf{blank} & \textbf{otherwise} \end{cases}$$

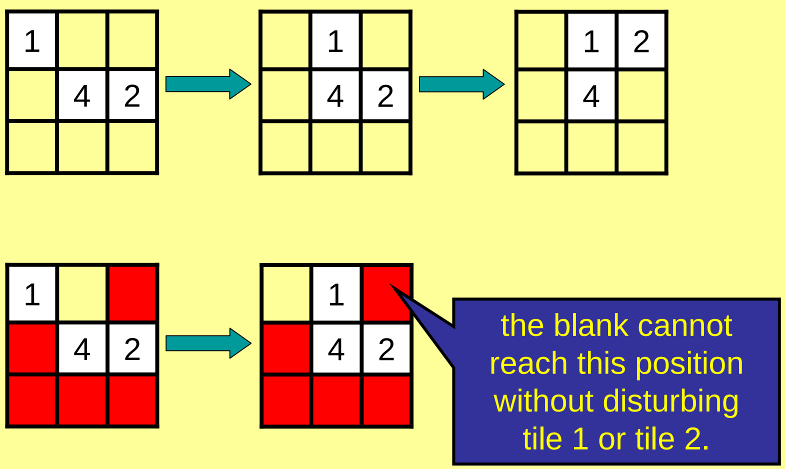
Korf & Felner's Method

Partition the tiles in groups, $G_1, G_2, \ldots G_k$

$$\varphi_i(x) = \begin{cases} \mathbf{x} & \textbf{if} \;\; \mathbf{x} \in G_i \\ \textbf{blank} & \textbf{if} \;\; \mathbf{x} = \textbf{blank} \\ \blacksquare & \textbf{otherwise} \end{cases}$$

Moves of ■ cost <u>zero</u>

What's the Difference ?



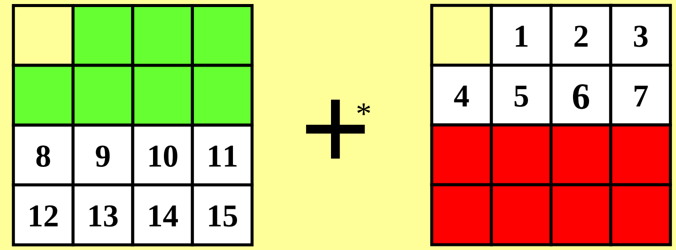
Compared to Max'ing

- If the PDBs were going to be max'd instead of added, we would count all the moves in all the PDBs.
- Therefore the PDBs for adding have smaller entries than the corresponding PDBs for max'ing.
- In initial experiments on the 15-puzzle, max'ing returns a higher value than adding for about 12% of the states.

Max’ing After Adding

8-7 Partition

(576 million entries)



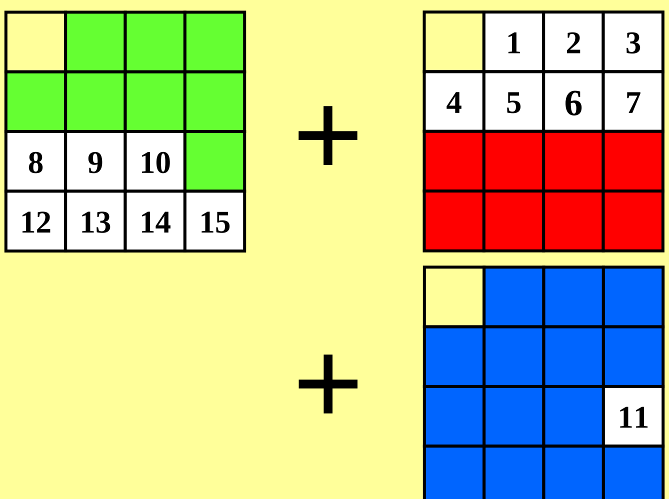
8 tiles retain their identity

7 tiles retain their identity

* movement of coloured tiles not counted

7-7-1 Partition

(115 million entries)



Max’ing after Adding

- For a given 7-7-1 partition, look up the 3 values and add them.
- Do this for each of the five 7-7-1 partitions and take the maximum*.

Also compute Manhattan Distance, and use that if it is largest of all. This was also done for 8-7.

15-Puzzle Results

Partition	n	Nodes Generated
7-7-1	5	**57,159**
8-7	1	**136,228**

58% reduction in #nodes generated,
but only 10% reduction in CPU time.

Customized PDBs

Space-Efficient PDBs

- Zhou & Hansen (AAAI, 2004)
  - Do not generate PDB entries that are <u>provably</u> not needed to solve the given problem.
  - Prune abstract state A if f(A) > U,
    where U is an upper bound on the solution cost <u>at the base level</u>.
- To work well, needs a heuristic to guide the abstract search and a fairly tight U.
- Even then requires significantly more memory than Hierarchical IDA*.

Reverse Resumable A*

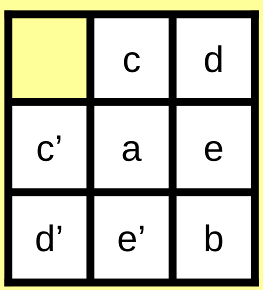
- Silver, 2005
- Aims to minimize the number of PDB entries
  - Backward search from abstract goal stops when abstract start is reached
  - If h(x) is needed and has not been computed, resume the abstract search until you get it.
- Requires abstract Open and Closed lists.

Super-Customization

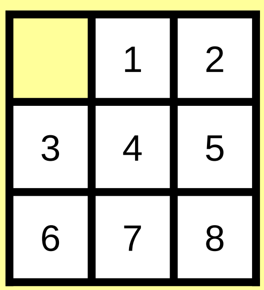
- If customizing an abstraction for a given start state is a good idea, wouldn't it be even better to change abstractions in the middle of the search space to exploit local properties ?
- This <u>does</u> pay off sometimes, even for PDBs:
  - Felner, Korf & Hanan (2004)
  - Hernadvolgyi (2003; also PhD thesis, chapter 5)

Multiple Lookups in One Pattern Database

Use Symmetries



mirror positions

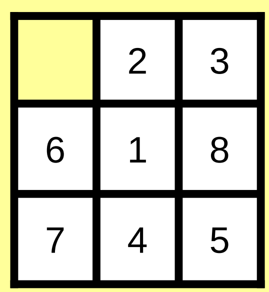


goal

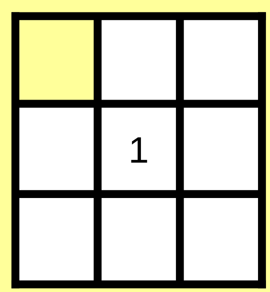
distance(Pos3,c') = distance(mirror(Pos3),c)

Example

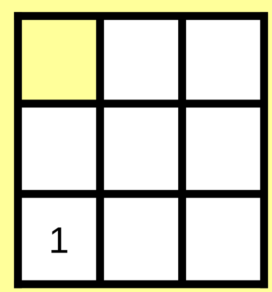
Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 1 □ □ □ □ □ □ □



state

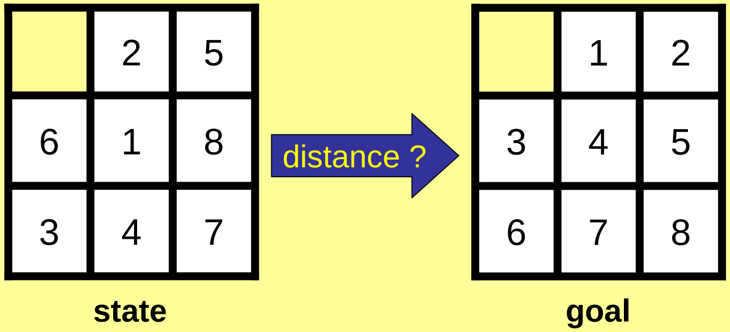


normal PDB lookup



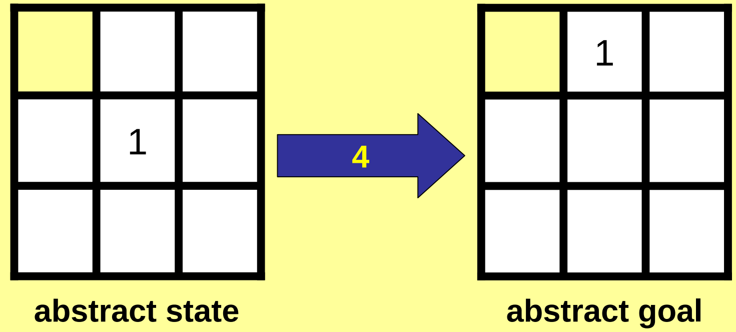
mirror lookup

“Dual” PDB Lookups

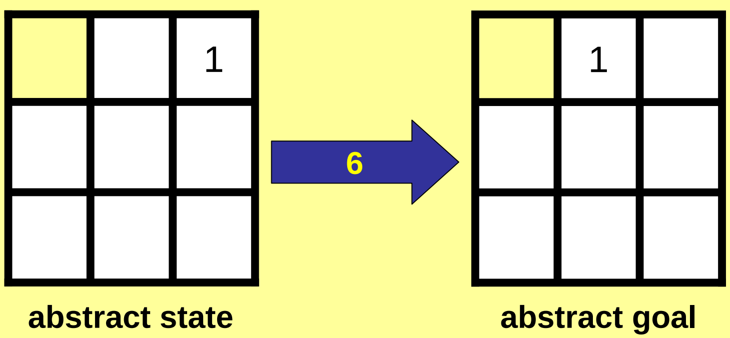


Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 1 □ □ □ □ □ □ □

Standard PDB lookup

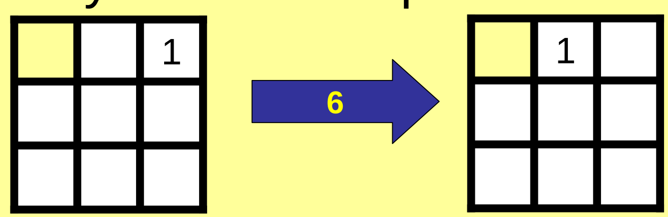


“Dual” lookup, same PDB

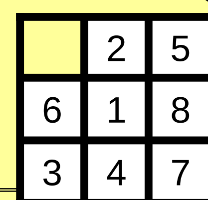


Relevance ?

Why is this lookup

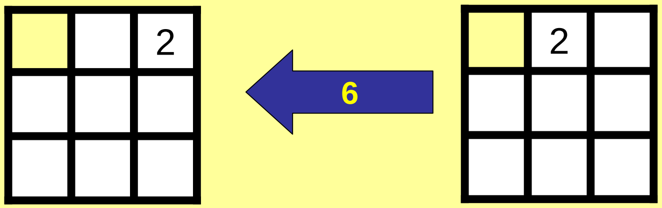


relevant to the original state ?



Tile 2

In a PDB for tile 2, this lookup

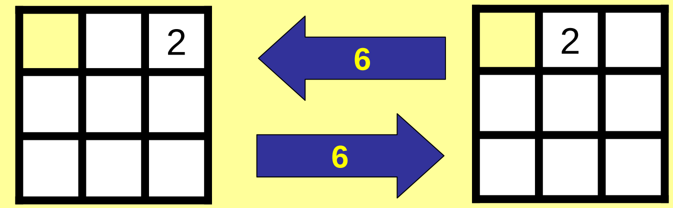


is relevant to the original state.

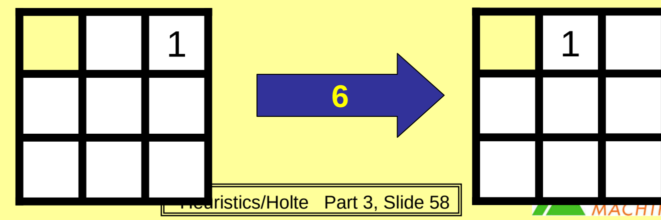
	2	5
6	1	8
3	4	7

Two Key Properties

(1) Distances are Symmetric



(2) Distances are tile-independent



Third Key Property

(3) Can determine which tiles in the given state correspond to the key tiles in the goal state.

Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 1 □ □ □ □ □ □ □

state

	2	5
6	1	8
3	4	7

goal

	1	2
3	4	5
6	7	8

Fourth Key Property

(3) The tiles that correspond to the key tiles in the goal state occur in the goal state.

Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 1 □ □ □ □ □ □ □

???

state

	999	5
6	1	8
3	4	7

goal

	1	2
3	4	5
6	7	8

Experimental Results

- 16-disk, 4-peg TOH, PDB of 14 disks
  - Normal: 72.61 secs
  - Only the “dual” lookup: 3.31 secs
  - Both lookups: 1.61 secs
- 15-puzzle, additive PDB (8-7)
  - Normal: 0.034 secs
  - Only the “dual” lookup: 0.076 secs
  - Both lookups: 0.022 secs

Dual Not Always Consistent

1	**2**	**3**	**4**	**5**	**6**	**7**	goal (7-pancake puzzle)

1	**2**	**3**	**4**	**6**	**7**	**5**
5	**7**	**6**	**4**	**3**	**2**	**1**

1	**2**	**3**				
1	**2**	**3**				

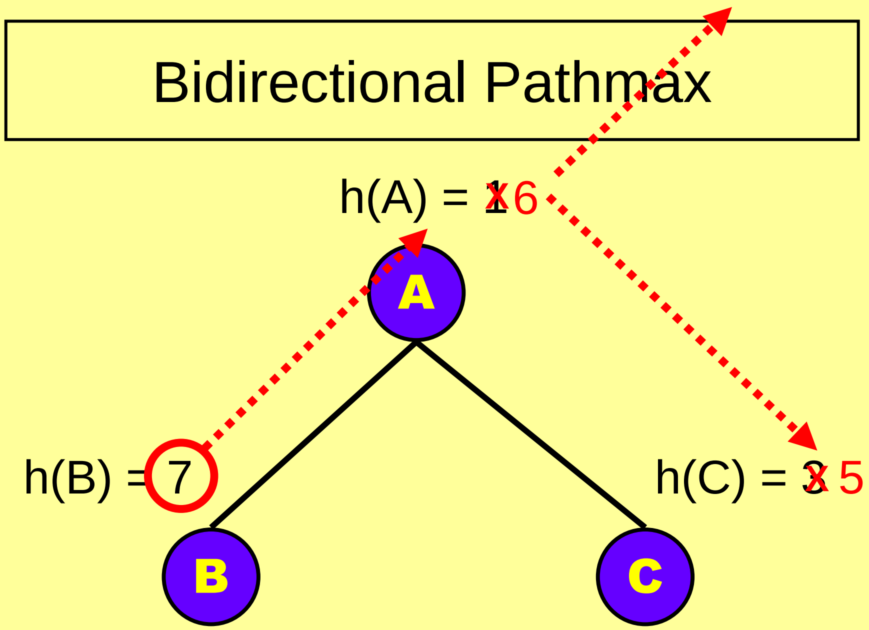
abstract distance = 0

				3	**2**	**1**
				1	**3**	**2**

abstract distance = 3

Domain = 1 2 3 4 5 6 7
Abstract = 1 2 3 □ □ □ □

Bidirectional Pathmax

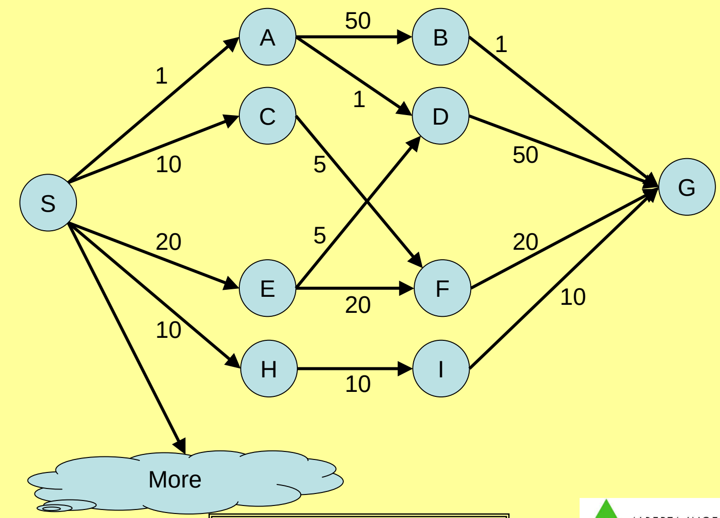


Related Algorithm – CFPD

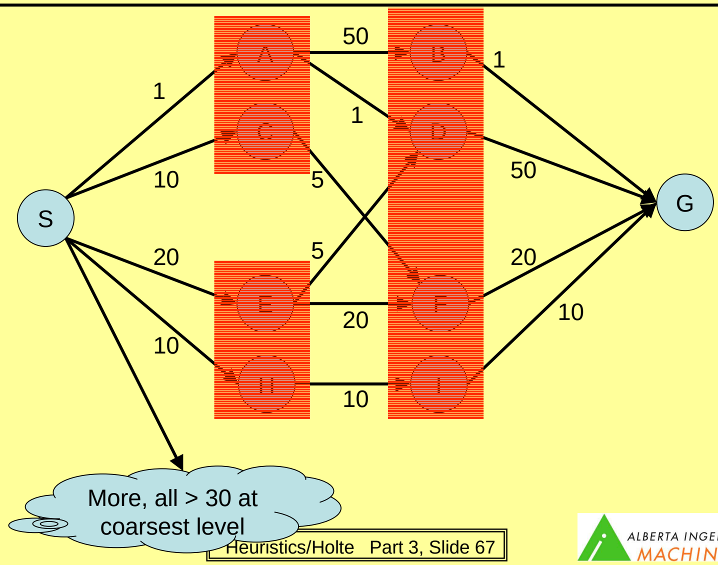
CFDP

- Coarse-to-Fine Dynamic Programming
- Works on continuous or discrete spaces.
- Most easily explained if space is a trellis (level structure).
- Abstraction = grouping states on the same level.
- Multiple levels of abstraction.
- Resembles refinement, but guaranteed to find optimal solution.
- Application: finding optimal convex region boundaries in an image.

CFDP - Example

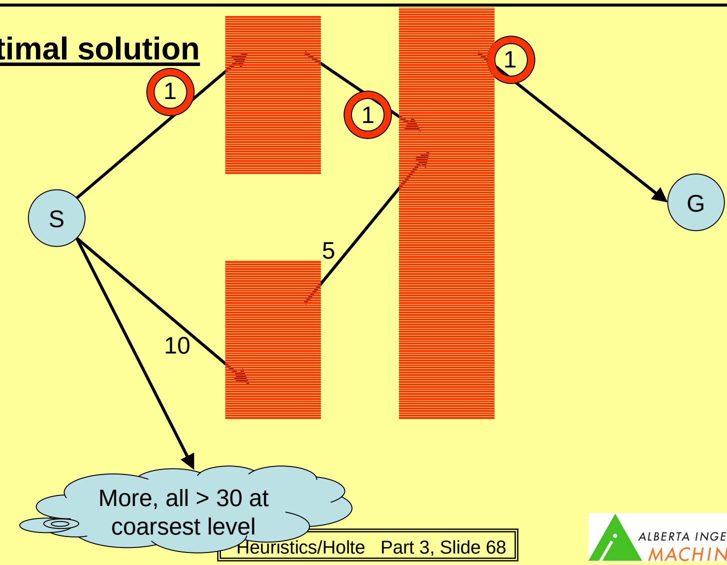


CFDP – Coarsest States

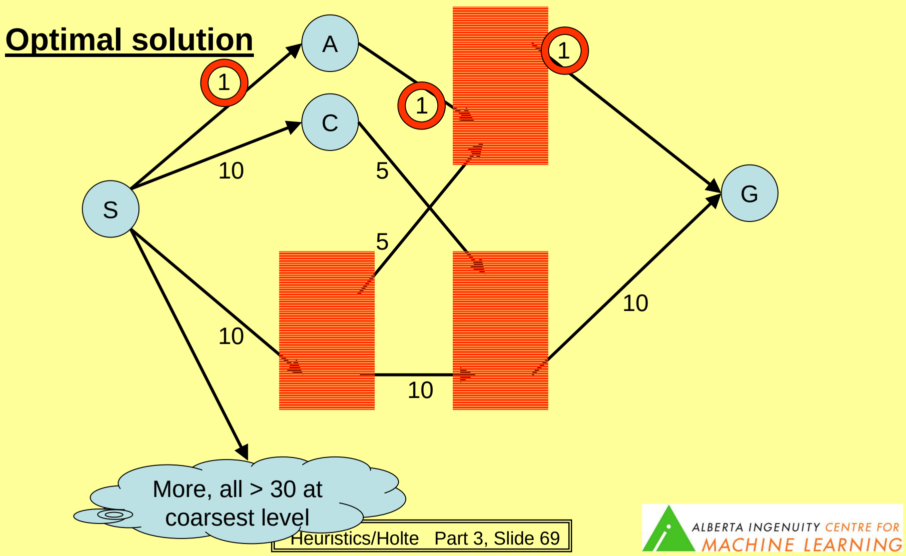


CFDP – Abstract Edges

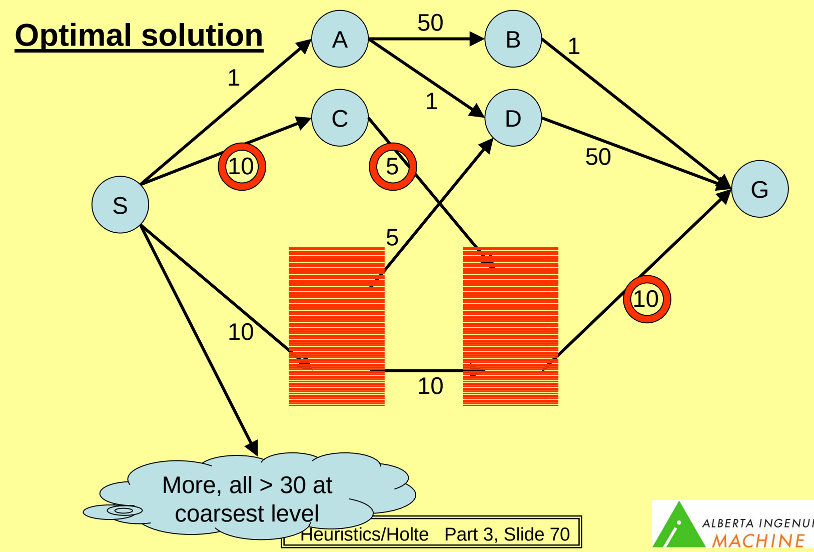
Optimal solution



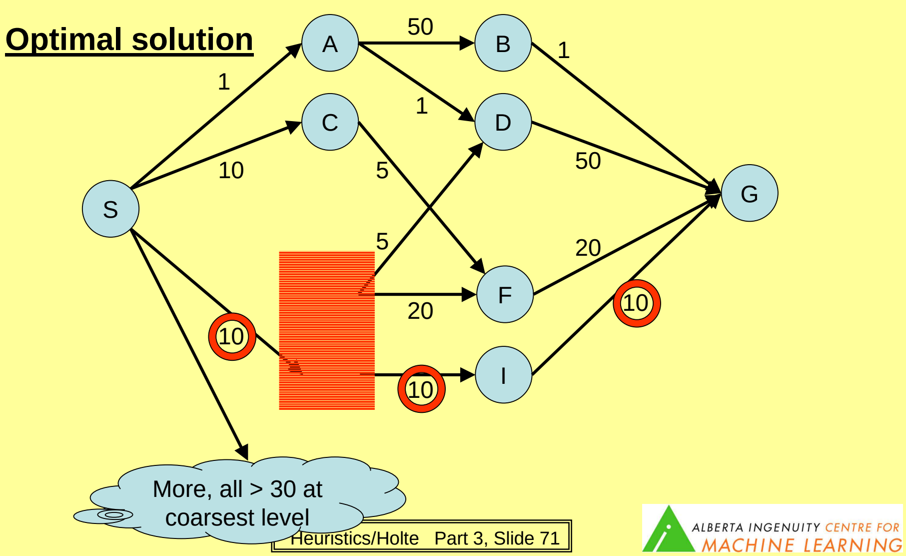
CFDP – Refine Optimal Path



CFDP – Refine Optimal Path



CFDP – Refine Again



CFDP – Final Iteration